Microsoft®
Windows™ 95

CURSO IBM DE PROGRAMACION

18

EN Windows 95

IBM

PLANETA DeAGOSTINI

Microsoft®
Windows™ 95

CURSO IBM DE PROGRAMACION EN Windows 95

VOLUMEN 2 - UNIDAD 18

Director Gerente: José Mª Puig de la Bellacasa
Director Técnico: Francisco Valer Bañales
Director de Marketing: Javier Graupera Gassol
Director Editorial: Artur Klein Moreno
Directora de Software: Beatriz Cerrato Vidal

Dirección y coordinación: José Luis Becerril Gasco. De la Corporación IBM España, Licenciado en Ciencias Físicas, Director del Instituto de Inteligencia Artificial de IBM España.

Redactores y colaboradores: *José Ramón Dorronsoro Ibero*. Doctor en Ciencias Matemáticas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid, Investigador del Instituto de Ingeniería del Conocimiento (UAM). *Vicente López Martínez*. Doctor en Ciencias Químicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Manuel Alfonseca Moreno*. Doctor Ingeniero de Telecomunicación y Licenciado en Informática, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Javier Garrido Salas*. Doctor en Ciencias Físicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Javier Martínez Rodríguez*. Doctor en Ciencias Físicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Santiago de Aréchaga Tarruell*. Ingeniero Superior de Informática, Investigador del Instituto de Ingeniería del Conocimiento (UAM). *Pedro Pascual Broncano*. Profesor Asociado del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Carlos Aguirre Maeso*. Licenciado en Ciencias Matemáticas, Profesor Asociado del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid, Investigador del Instituto de Ingeniería del Conocimiento (UAM).

Realización editorial: LH servicios informáticos, S.L.

Diseño gráfico: Esteoeste

© Multimedia Ediciones, S.A., Barcelona
ISBN Obra Completa: 84-88720-79-3
ISBN Fascículos: 84-88720-80-7
ISBN Volumen 2: 84-88720-82-3
Depósito Legal: B-33.798-97

Redacción y administración:
Aribau 198, 5ª planta - 08036 - Barcelona
Tel. (93) 306.78.90 - Fax (93) 306.78.97

Servicio de atención al cliente: Teléfono 902.101.324

Petición de números atrasados:
Teléfono 902.288.288
Apartado de correos nº 2.867, -08080- Barcelona

Servicio de consulta:
Teléfono 902.244.444 de lunes a viernes, de 10.00 a 14.30 y de 15.30 a 19.00 horas.
Fax (91) 661.67.08 durante las 24 horas del día.

Distribuye para España: MIDESA. Marco Ibérica de Ediciones, S.A., Carretera de Irún, km. 13,350, variante de Fuencarral - 28034 - Madrid

Venta en quioscos y librerías: Pida a su proveedor que le reserve sus ejemplares del curso. Adquiriéndolos todas las semanas en el mismo quiosco o librería facilitará la distribución y obtendrá un mejor servicio. El editor se reserva el derecho de modificar el precio de venta de los componentes de la colección en el transcurso de la misma, si las circunstancias del mercado así lo exigieran.

Composición del Curso: El Curso IBM de Programación se publica en forma de 52 unidades de aparición semanal y 52 CD-ROM. Las 52 unidades se encuadernan en 5 volúmenes. Con la unidad que completa cada volumen se ponen a la venta las tapas para su encuadernación.

Multimedia Ediciones garantiza la publicación de todos los elementos que componen esta obra.

© Editorial Planeta Argentina S.A.I.C.
Independencia, 1668 - Capital Federal
Teléfono atención al cliente: 482-0655 y 482-0657
Distribuye en capital: Huesca-Sanabria. En interior: D.G.P.
© Editorial Planeta Mexicana S.A. de C.V.
Avda. Insurgentes # 1162 - México D.F.
Teléfono de atención al cliente: 575-1348
© Editorial Planeta Colombiana S.A.
Carrera 68 A, nº 25-55 - Santafé de Bogotá
Teléfono de atención al cliente: 9800-15315

Imprime: Cayfosa. Carretera de Caldes, km. 3, Santa Perpetua de Mogoda (Barcelona).

INSTALACION/DESINSTALACION DEL SOFTWARE: Para la correcta instalación y desinstalación del CD-ROM que se entrega en esta unidad, véase el libreto que acompaña a cada CD-ROM.

Marcas comerciales registradas: En esta publicación son marcas registradas de sus respectivas compañías: Delphi Desktop, dBase, Turbo Pascal, Turbo C++, Turbo Assembler, Turbo Debugger, Borland Intrabuilder, Paradox, Visual dBase y BRIEF (Borland International, Inc.). Lotus, Lotus 123 y SmartIcons (Lotus Development Corporation, filial de IBM). Microsoft DOS, Microsoft Windows, Microsoft Windows 95, Excel, Visual Basic, Access y FoxPro (Microsoft Corporation). Netscape Navigator Gold (Netscape Communications Corporation). Java (Sun Microsystems, Inc.). IBM (International Business Machines).

CURSO DE PROGRAMACIÓN VISUAL: TURBO PASCAL (5)

DOMINANDO LA MEMORIA DEL ORDENADOR

Los registros permiten al programador definir estructuras de datos "a la medida" de los elementos que forman parte del problema que está resolviendo con la aplicación. Al concluir esta unidad estará preparado para escribir aplicaciones sencillas empleando el lenguaje Pascal.

En los registros se integran tipos de datos distintos para formar una unidad conceptual que facilite la programación de los algoritmos que intervienen en una aplicación. Pero, antes de estudiar los registros, hay que conocer las **enumeraciones** y los **conjuntos**, que son estructuras del Pascal que también agrupan datos sencillos.

Los **archivos** se utilizan para almacenar datos y su manejo desde el programa es necesario en muchísimas aplicaciones. En esta unidad se estudiarán los pasos que se han de dar en un programa para guardar información en un fichero y recuperarla posteriormente. Las variables **"punteros"** que almacenan direcciones de variables constituyen un elemento fundamental de todos los procedimientos de programación avanzada. Son las que permiten gestionar dinámicamente la memoria reservada para el programa.

Todos estos conceptos de programación son los que se estudiarán en esta unidad. También se abrirán las puertas que permiten entrever Turbo Pascal como una herramienta para desarrollar aplicaciones profesionales en el entorno gráfico Windows.

La aplicación Estrategia, que se suministra con todo el código fuente, es un ejemplo que permite indagar sobre la programación avanzada de aplicaciones.

Para comenzar el itinerario por todos estos nuevos conceptos de programación, se ha elegido el más sencillo: las enumeraciones.

INSTALACIÓN

La instalación de la unidad es muy sencilla y mantiene los mismos pasos que en entregas anteriores. Primero hay que insertar el CD en el lector del equipo y después, abrir el menú Inicio y activar la opción Ejecutar. Suponiendo que el lector CD fuera la unidad D: del ordenador, hay que teclear: D:\INSTALAR

En caso de tener activada opción AutoRun de Windows 95, aparecerá la ventana de instalación sin necesidad de teclear ninguna orden. A partir de aquí, solamente habrá que elegir en la ventana el disco donde se desee ubicar Turbo Pascal y sus componentes. El programa de instalación copiará desde el CD los archivos necesarios, que se disponen en el directorio \TPW del disco elegido.

ENUMERACIONES

Después de haber escrito varios programas, cualquier lector se habrá dado cuenta de que en muchas ocasiones se utilizan los números 1, 2, 3, etc., para representar objetos abstractos a los que le resulta muy conveniente esa ordenación numeral.

Por ejemplo, es razonable emplear números del 1 al 7 para referirse a los días de la semana, o del 1 al 4 para las estaciones del año. Pues bien, pensando en estos casos, los diseñadores del Pascal incluyeron las **enumeraciones**, que son una forma cómoda de asociar a secuencias de valores (números ordinales) secuencias de palabras (indicadores).

Para entenderlo, lo mejor es ver un caso concreto. Por ejemplo, para referirse a los meses del año en un programa, se puede establecer una relación entre un número y el mes, empezando con el 1 para Enero y siguiendo hasta el 12 para Diciembre, y con la ordenación del calendario. Dado el uso cotidiano de esta relación, resultaría muy fácil escribir y leer programas en los que se utilizara una variable "mes" del tipo **Byte**. Siempre que "mes" tomara el valor 2, el programador inmediatamente interpretaría ese valor como el mes de Febrero. Pero si en un programa se quiere operar con datos del tipo color, considerando las posibilidades Rojo, Amarillo, Naranja, Verde y Azul, no será fácil elegir una ordenación entre ellos que permita establecer una relación que se recuerde fácilmente entre un número (por ejemplo del 0 al 4) y cada color.

Precisamente para estos casos son útiles las enumeraciones, ya que permiten establecer esa relación en la definición de un tipo de dato y el programador puede olvidarse de dicha relación durante la escritura del programa.

Un tipo de dato "enumerado" se especifica con el conjunto de posibilidades entre paréntesis y separadas por comas. Por ejemplo, en el programa "Colores" que aparece en la figura 1, se puede ver la definición de una enumeración de cinco valores asociados a cinco nombres de colores. El tipo de dato "Color" se define como la enumeración de esos cinco colores. La variable "i" se declara del tipo "Color" y, por tanto, puede tomar el valor

"Amarillo". El compilador establece una relación unívoca entre cada uno de los posibles valores de la enumeración y un número, que será el de orden con el que aparezca cada valor en la declaración, empezando con el 0 para el primero.

Sobre un tipo de dato declarado como enumeración pueden operar funciones que admiten argumentos ordinales, entre otras "Succ" y "Pred" (siguiente elemento y anterior elemento).

Los indicadores de los valores enumerados no se pueden leer o escribir. Sólo existen para el compilador y desaparecen en tiempo de ejecución; durante éste, únicamente permanecen los números ordinales que están asociados a cada elemento de la enumeración. Por ejemplo, la instrucción "ReadLn(i);" no sería válida en el programa "Colores".

Sin embargo, se puede leer un número ordinal que luego se transforma al valor de la enumeración mediante el indicador del tipo de dato y, entre paréntesis, el valor ordinal. En la figura 2, una modificación del programa "Colores" lee un número de orden con "ReadLn(n)" y lo transforma en uno de los valores de la enumeración (con "i:=Color(n)").

TRUCO

El procedimiento "DoneWinCrt" que se llama al final del programa "Colores", está declarado en la unidad WinCrt y sirve para cerrar la ventana de Windows que se ha abierto para permitir el intercambio de datos. De este modo, se evita que quede inactiva, pero abierta, una vez que haya finalizado la ejecución del programa.

Combinada con "ReadLn" consigue que, cuando el usuario pulse la tecla Intro, se cierre la ventana una vez que ha acabado la ejecución.

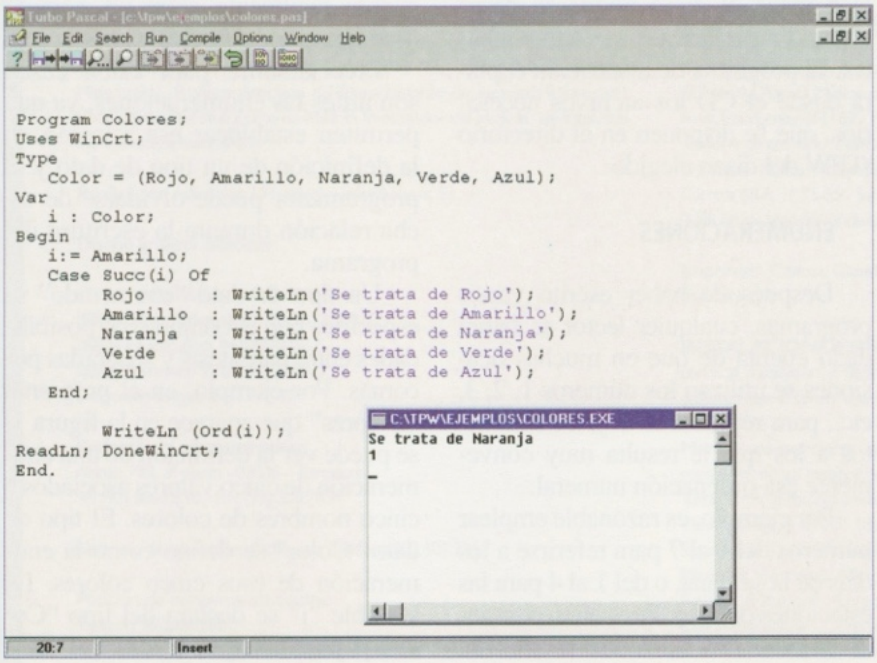


Fig. 1. La enumeración permite recordar fácilmente valores de números ordinales cuando se quieren asociar a nombres.

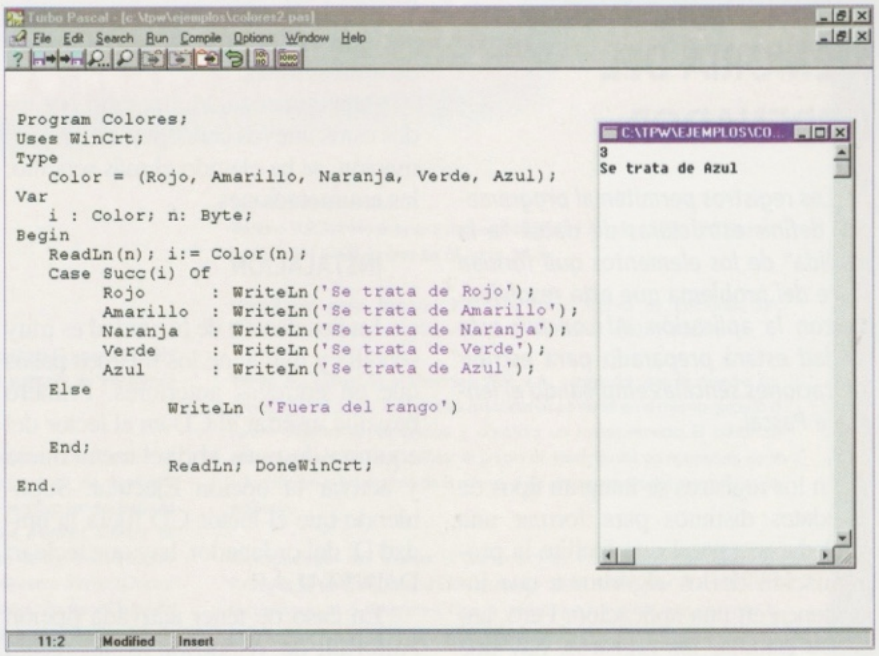


Fig. 2. Los nombres de los elementos de una enumeración no pueden leerse directamente.

CONJUNTOS

Un conjunto, como su nombre sugiere, es un grupo de datos que tienen una asociación en la que es irrelevante el orden de los elementos.

En Pascal también se pueden representar conjuntos, siempre y cuando los datos que reproduzcan sus elementos sean números ordinales. También es posible que sean elementos del conjunto los definidos con una enumeración. Un conjunto cuyos componentes fuesen del tipo "Cualquiera", se definiría con la siguiente sintaxis:

```
Type identificador =
Set Of Cualquiera
```

Fig. 3. Sintaxis de la definición de un conjunto.

donde "identificador" sería el nombre elegido para el nuevo tipo de dato que representa un conjunto.

La especificación de los miembros de un determinado conjunto se realiza escribiendo los elementos, uno a uno, o mediante rangos, entre paréntesis cuadrados.

tenece a un conjunto, con la siguiente sintaxis:

```
NomElemento In
NomConjunto
```

Fig. 5. Sintaxis del operador In.

donde "NomElemento" es el valor del elemento sobre el que se indaga y "NomConjunto" el indicador del conjunto. El resultado de esta operación es un dato **Boolean** y se evalúa a **True** sólo si el elemento pertenece al conjunto.

OPERADORES ACTUANDO SOBRE CONJUNTOS

Si "A", "B" y "C" son conjuntos de elementos del mismo tipo, los operadores aritméticos producen los siguientes resultados:

```
C := A + B        {C : conjunto unión de A y B}

C := A * B        {C : conjunto intersección de A y B}

C := A - B        {C : conjunto diferencia de A y B}
```

También se pueden comparar los conjuntos con los operadores lógicos dando lugar a un dato **Boolean**. Si "verdad" es una variable del tipo **Boolean**, las comparaciones siguientes serán posibles:

```
verdad := A  =  B  {verdad es True si A y B son iguales}

verdad := A á> B  {verdad es True si A y B son distintos}

verdad := A á= B  {verdad es True si A es subconjunto de B}

verdad := A => B  {verdad es True si B es subconjunto de A}
```

teniendo en cuenta que el conjunto vacío es subconjunto de todo conjunto. El conjunto vacío se indica con "[ ]". El número de elementos que pueden formar un conjunto en Pascal está limitado a un máximo de 256.

REGISTROS

En esta sección se estudiará uno de los conceptos más prácticos en un lenguaje moderno de propósito general. Se trata de los registros, un tipo de dato con estructura que puede definir el programador ajustándose a sus necesidades. En un registro se pueden mezclar datos de distinto tipo.

Por tanto, es posible asociar a un objeto con estructura todo tipo de información, por muy heterogénea que sea.

Por ejemplo, se trata de escribir una aplicación en la que se procese información referente a alumnos de un curso: se quiere representar el concepto "Alumno" por un conjunto de información que lo identifique, como pueden ser su nombre, apellidos y las calificaciones en los distintos exámenes teóricos y ejercicios prácticos.

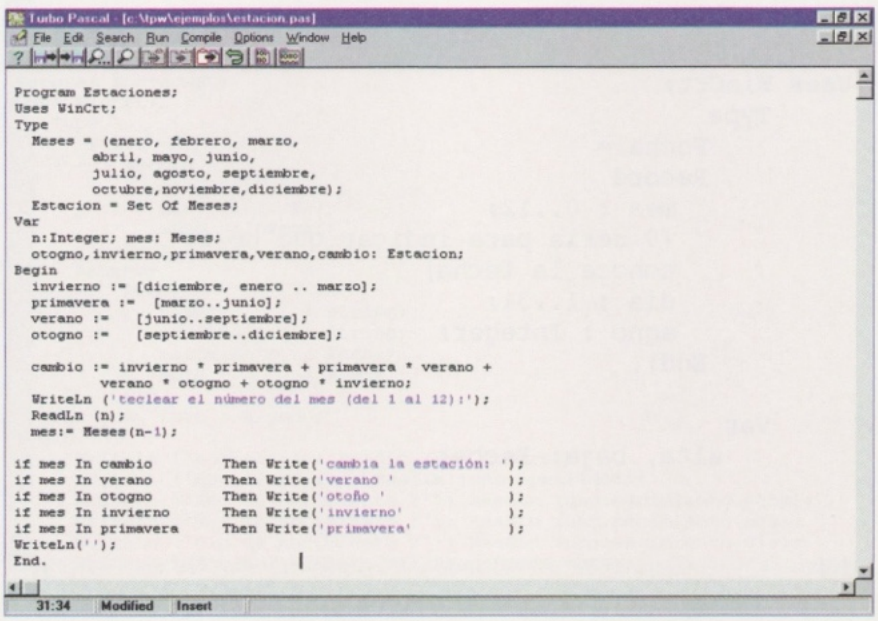
Fig. 4. En el programa "Estaciones" aparece un ejemplo de la declaración de conjuntos, así como del manejo de operadores con los mismos.

EL PROGRAMA "ESTACIONES"

En el principio del programa "Estaciones", que puede verse en la figura 4, se muestra un ejemplo de la definición de un tipo de conjunto, el tipo "Estacion".

Para el programa se definen 5 variables que son conjuntos del tipo "Estación" y cuyos elementos son del tipo "Meses". En este caso, los elementos están declarados en una enumeración. Se trata de los nombres de los meses del año ordenados de modo que, internamente, Enero se representará por un 0 y Diciembre por un 11.

Pascal también permite operar sobre conjuntos. El operador **In** se utiliza para determinar si un elemento per-



Es decir, se necesita un tipo de dato "Alumno" en el que se pueda "encapsular" información estructurada en "campos" como el nombre, apellidos, notas y otros. A un tipo de dato definido de este modo por el usuario se le denomina **registro (Record)** y a cada uno de sus componentes, **campo**. El programador da nombre al nuevo tipo de dato y a cada uno de los campos que lo componen y especifica el tipo de dato que puede ocupar cada uno de los campos. La sintaxis para estas especificaciones es:

```
Type
 nombre =
  Record
   NombreCampo1 : TipoDato1;
   NombreCampo2 : TipoDato2;
    . . . . . .
  End;
```

Fig. 6. Sintaxis de especificación de un registro.

donde "nombre" es el identificador elegido para ese tipo de registros; "NombreCampo1", "NombreCampo2",..., los identificadores elegidos para los distintos campos que componen el registro; y "TipoDato1", "TipoDato2", ..., las especificaciones del tipo de dato que se va a almacenar en cada uno de los campos.

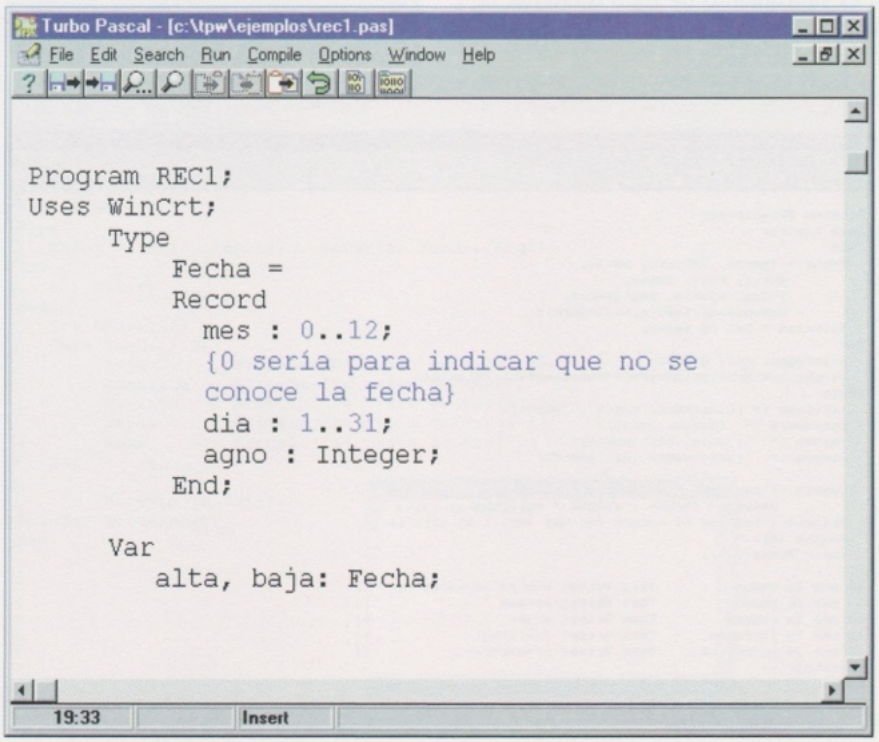
Hay casos en los que puede ser conveniente definir un tipo de variable para almacenar la información de una fecha. Una posible declaración de registro apropiada para ello es la que puede verse en el principio del programa "REC1" en la figura 7.

El registro engloba los campos y, para referirse al valor de un campo del registro, se utiliza el nombre de la variable del tipo registro seguido de un punto y del nombre del campo.

En la figura 8 puede verse una posible continuación del programa REC1. En él se utilizan dos variables del tipo "Fecha", que son registros con estructura interna. El programa realiza lo siguiente:

- En la primera asignación, se guarda el valor 8 en el campo "mes" de la variable "baja".
- En la siguiente instrucción, se guarda en el campo "mes" de la variable "alta" el valor que se almacenaba en el mismo campo de la variable "baja" incrementado en 2.
- El resto de las asignaciones pueden entenderse sin dificultad.

Fig. 7. "Fecha" es un tipo de dato con estructura interna. Es un registro con tres campos.



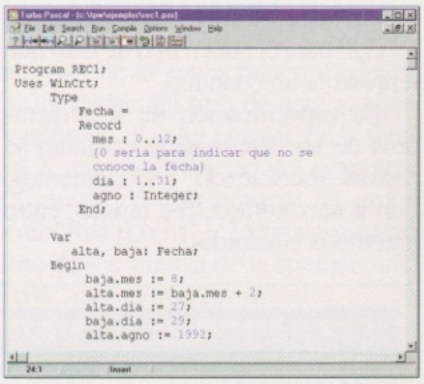


Fig. 8. Los campos de las variables que son registros se identifican por el nombre de aquéllas, separados con un punto del nombre del campo.

• La estructura With

Pascal permite la comodidad de la estructura **With** para acceder a los valores de los campos de los registros, sin tener que repetir demasiadas veces el nombre de la variable. La sintaxis de esta estructura es la siguiente :

```
With nombre Do accion
```

donde "nombre" es el identificador de la variable (o las variables separadas por comas), que es un registro y a la que se refiere la instrucción "accion" que se encuentra a continuación del **Do**. Si se trata de una instrucción compuesta, se encontrará como siempre horquillada entre un **Begin** y un **End**.

En la instrucción **With** se pueden especificar los distintos valores del registro utilizando sólo el nombre de los campos.

En la figura 9 se usa la estructura **With** en el programa "REC2" para evitar repetir el nombre de la variable "baja". Si aparece dentro de la instrucción el nombre aislado del campo "mes", se refiere al campo "mes" de la variable "baja".

Un registro no sería útil si para referirse a toda la información almace-

nada en él se tuviese que hacer referencia siempre a cada uno de los campos. Este paso no es necesario, ya que pueden asignarse de golpe los valores de todos los campos de un registro. Por ejemplo, con

```
alta := baja;
```

se rellenan de golpe todos los campos del registro "alta" con los valores que almacena para todos sus campos el registro "baja".

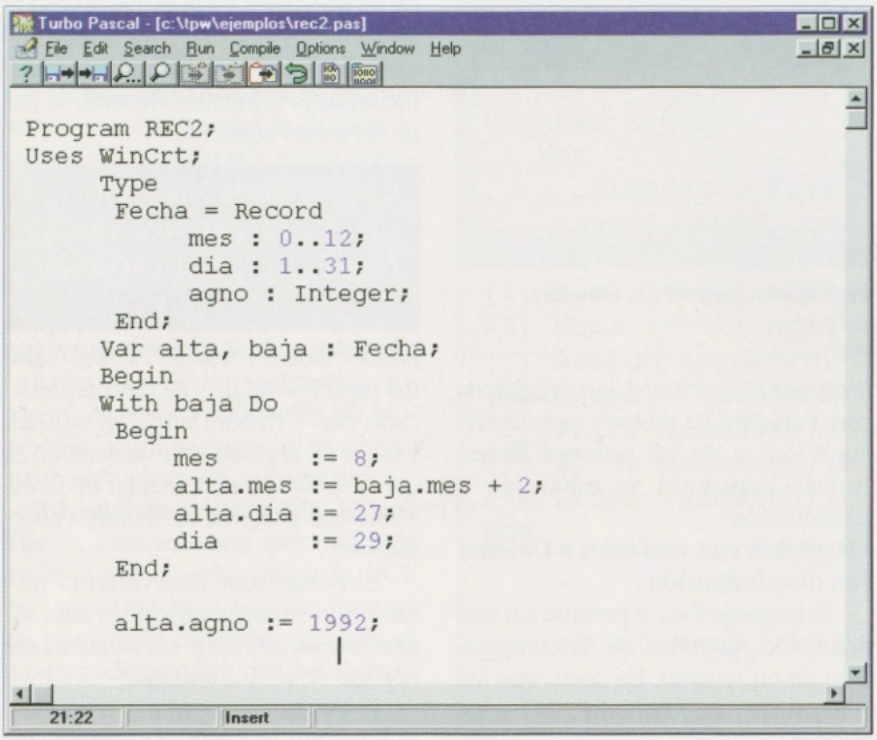


Fig. 9. La estructura **With** tiene que servir para hacer más legibles los programas que utilizan variables que son registros.

• Registros de registros, Arrays de registros

Los campos de un registro pueden contener, a su vez, datos con estructura de registro. Si se amplia el programa "REC2" hacia la construcción de una base de datos con los alumnos y sus notas, un programa "EntradaBD" de captación de datos podría empezar tal como lo puede verse en la figura 10. En ese programa se hace referencia a los campos de registros que son a su vez campos de un registro, lo que se logra concatenando con puntos los identificadores de los nombres de los campos: "uno.nacimiento.mes" identifica el valor del campo "mes" del campo "nacimiento" de la variable "uno".

Para crear una base de datos que contenga información de los alumnos de una clase, lo lógico sería ordenar los alumnos en una lista. Para ello, se puede crear una estructura **Array** cuyos elementos sean los registros definidos para almacenar la información de los alumnos. Se podría, por ejemplo, añadir en la parte declarativa del programa "EntradaBD" la siguiente declaración de tipo de dato:

```
Lista = Array [1..100]
of alumno;
```

Fig. 11. Declaración de la lista de alumnos.

y, si se declara la variable "primero" del tipo "Lista", en ella se podrá guardar la información de todos los alumnos del curso.

El acceso a los elementos del **Array** es el habitual. Por ejemplo, y tomando como referencia el programa "EntradaBD", se tendría que con

```
primero[1] := uno;
```

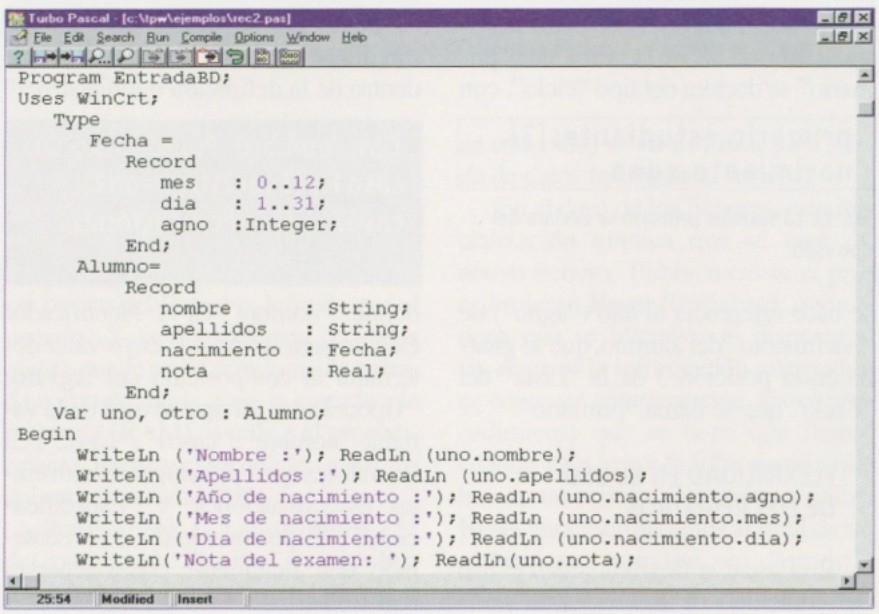
se asigna al primer elemento de la variable "primero" tipo "Lista", de golpe, todo el contenido de la variable "uno" del tipo "alumno".

Para referirse a un campo de los registros organizados en forma de **Array**, se utilizan los puntos como separadores. Con

```
Primero[3].nacimiento.mes
```

se hace referencia al valor almacenado en el campo "mes" del campo "nacimiento" del dato del tipo "Alumno" almacenado en la posición tercera de la "Lista" "primero".

Fig. 10. Los campos de los registros pueden ser también, a su vez, registros.



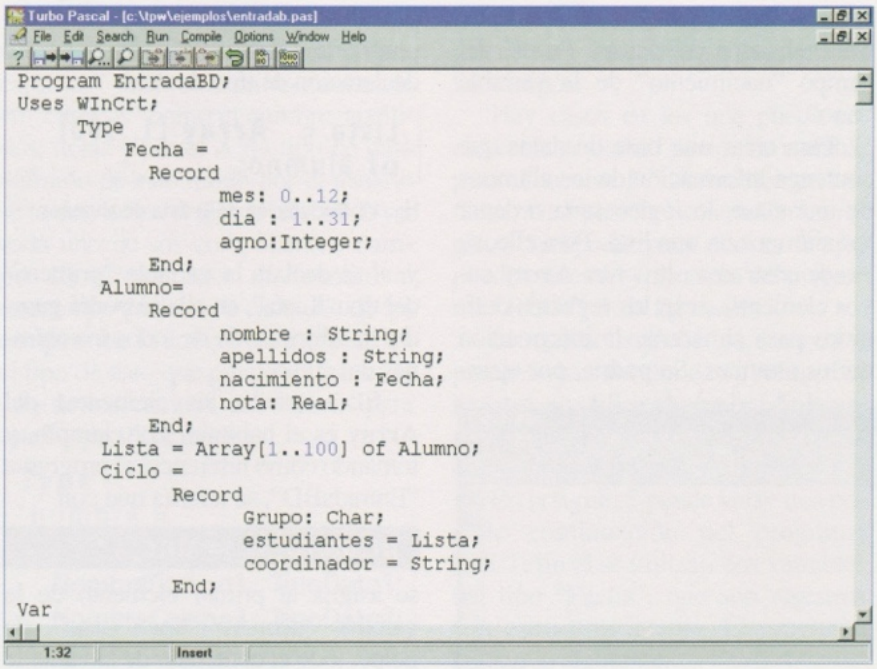


Fig. 12. El programador puede declarar variables que se ajusten bastante a la estructura conceptual de la información que van a almacenar.

• Registros cuyos campos son del tipo Array

También existe, obviamente, la posibilidad inversa: la de definir registros en los que sus campos sean datos con la estructura de **Array**. En la figura 12 puede verse la ampliación del programa "EntradaBD" en la que se declara el tipo de dato "Ciclo", que es un registro con tres campos, donde uno de ellos es del tipo **Array**: el denominado "estudiantes" es una "Lista", es decir, un **Array** de datos del tipo "alumno". Si la variable "primario" se declara del tipo "ciclo", con

```
primario.estudiantes[3].
nacimiento.agno
```

Fig. 13. La variable primario se declara del tipo ciclo.

se hace referencia al año ("agno") de "nacimiento" del alumno, que se guarda en la posición 3 de la "Lista" del "Ciclo" que se llama "primario".

FLEXIBILIDAD EN EL USO DE LOS REGISTROS

En Pascal la flexibilidad en el uso del contenido de los registros viene dada por la opción de los **registros con variante** (o uniones con discriminación) y de las **uniones libres**. Ambas estructuras se estudiarán a continuación.

• Registros con variantes o Uniones con discriminación

El lenguaje Pascal permite dar una definición dinámica de la composición de un registro. Es decir, que los campos que lo componen varíen de un lugar a otro del programa.

Para ello, se utiliza la siguiente modificación de la estructura **Case** dentro de la definición del registro

```
Case nombre: TipodeDato Of
caso1 : (Especificacion1);
caso2 : (Especificacion2);
. . . . . .
```

donde: "nombre" es el identificador elegido para la variable cuyo valor determina la composición del registro; "TipodeDato", el tipo de dato de la variable "nombre"; "caso1", "caso2", ..., los valores que darán lugar a las distintas especificaciones; y "Especificacion1", "Especificacion2", ..., el contenido que finalmente podrá tener el registro.

La enumeración de las opciones no acaba con un **End**; porque existe la restricción de que las variantes se han de colocar en la última parte de la definición de un registro y, por tanto, acaban con el **End**; del fin de la definición del mismo.

• Uniones libres

Con las Uniones libres se hace más explícito que la utilización de los campos alternativos es pura responsabilidad del programador y no se indica identificador para la variable que gobierna las opciones. Para especificar esta alternativa dentro de la definición de la estructura **Record**, se hace del modo siguiente:

```
Case TipodeDato Of
caso1 : (Especificacion1);
caso2 : (Especificacion2);
. . . . . .
```

donde "caso1", "caso2", ..., son valores posibles del tipo de dato especificado con "TipodeDato". Es equivalente a un registro con variante en el que se omite el identificador de la variable que gobierna las distintas alternativas.

El compilador reserva en la memoria un espacio suficiente para almacenar en ella la de las variantes del registro de mayor tamaño.

El espacio de memoria se rellenará tal como lo desee el programador en cada momento y la elección se realizará usando un campo u otro. Igualmente, la interpretación de los bits almacenados en esas posiciones de memoria también dependerá del campo del registro que se utilice, en el caso de que distintos campos correspondan a diferentes tipos de datos.

No hay que olvidar que, tanto las uniones libres como los registros con variantes, exigen que sea el último campo de un registro el único que admita la variabilidad.

Las **Uniones Libres** han de usarse con precaución para evitar confusiones en la lectura de los programas pero, a veces, son las adecuadas para hacer un programa legible.

MANEJO DE FICHEROS

La entrada y salida de datos desde un programa no tiene porqué realizarse utilizando los dispositivos estándar de entrada y salida (teclado y terminal), sino que puede realizarse a través de cualquier periférico.

También es posible, y de hecho es lo más frecuente, utilizar los dispositivos de almacenamiento intermedio de datos. Estos dispositivos son discos magnéticos u ópticos en los que diferentes trozos contienen información que se agrupa en archivos (o ficheros). Un archivo (o fichero) es la fracción lógica en la que se basa la división de toda la información que puede contener un disco duro, un disquete, un CD-ROM, u otro dispositivo de este tipo.

• Ficheros con Tipo

Para poder referir todas las entradas y salidas, el lenguaje Pascal utiliza un tipo de dato que se denomina **File** (Archivo). Con él se pueden direccionar las entradas y salidas de los programas a impresoras, dispositivos auxiliares y, también, a los archivos.

El valor de una variable del tipo **File** es esencialmente una dirección hacia la que se debe dirigir la UCP para transferir datos. El valor concreto de la variable es irrelevante para el programador, puesto que las instrucciones que a ella se refieren nunca requieren conocerlo.

Pascal provee un procedimiento que permite asignar a una variable del tipo **File** la dirección del dispositivo de entrada o salida que el programador pretende utilizar. Éste es el procedimiento **Assign** (Asigna) que tiene dos argumentos:

- El primero es el nombre de la variable definida del tipo File
- El segundo es un dato del tipo String que contiene el nombre con el que el sistema operativo identifica el fichero que se quiere utilizar.

En la instrucción {3} del programa "Escribe", que puede verse en la figura 14, se especifica que con la variable "almacen" se hace referencia al archivo que el sistema operativo reconoce como c:\tpw\escribe.sal. La variable "almacen" se ha declarado como **File of Byte**. En la declaración se especifica que los datos que se van a leer o escribir en ese dispositivo son del tipo **Byte**.

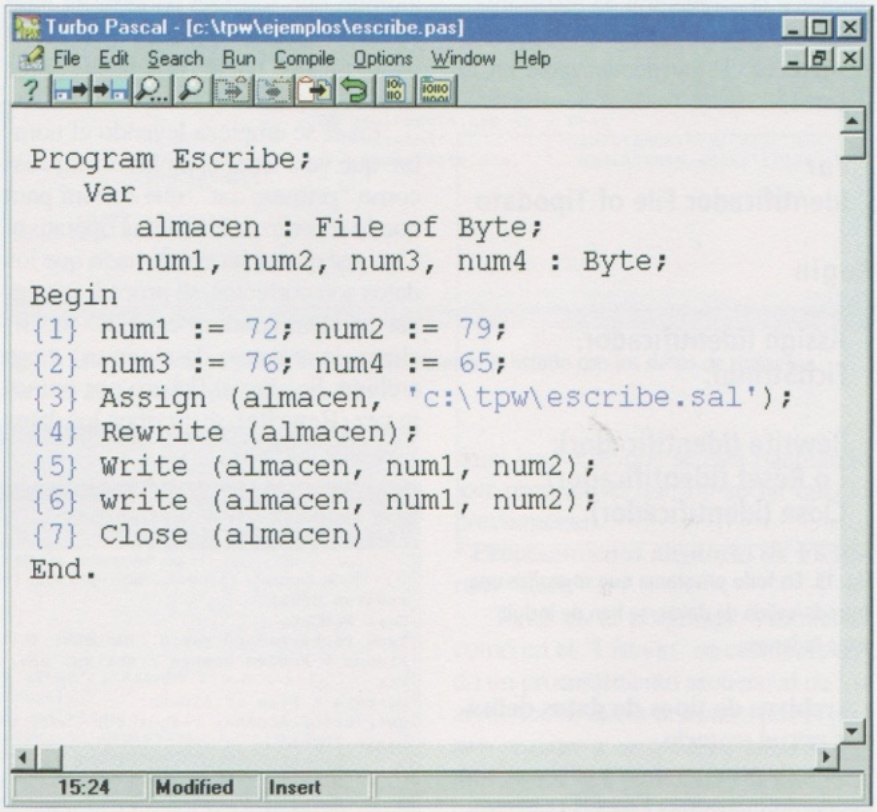


Fig. 14. "Almacen" es el indicador que mantiene la información sobre el archivo en el que se quieren escribir los datos.

• Procedimientos para el manejo de ficheros

El tipo de dato que va a intercambiarse con un dispositivo especificado con una variable del tipo **File**, puede ser básico del Pascal o definido por el usuario. Y la transferencia de datos que se puede realizar es tanto de entrada como de salida. Para la entrada a la memoria (RAM) se utiliza el procedimiento **Read** y para salida (a los archivos) se utiliza **Write**.

Por ejemplo, en la instrucción {5} del programa "Escribe" se escriben los valores almacenados en "num1" y "num2" en el periférico identificado por "almacen", es decir, en el archivo c:\tpw\escribe.sal.

En el programa "Escribe" también aparecen las llamadas a dos procedimientos: **Rewrite** (Re-escribe) y **Close** (Cierra), relacionados con las tareas que se realizan en un ordenador para transferir datos.

Con el primero, **Rewrite**, se especifica que el dispositivo ("almacen", en este caso) se va a utilizar para salida de datos (escritura al fichero).

En el caso de los ficheros, esta inicialización implica que se cree un nuevo fichero. También existe el procedimiento **Reset** (Inicializa), que ordena que se inicialice el dispositivo sin destruir la información que pudiera contener anteriormente. Es el procedimiento que se tiene que llamar cuando va a leerse la información que hay en un archivo. También se utiliza **Reset** para escribir datos al final de un archivo que ya existe, sin destruir la información anterior.

En la instrucción {7} del programa "Escribe" se procede a realizar la liberación del archivo de control ejercido por el programa con el procedimiento **Close**, que cierra el canal de comunicación abierto anteriormente con el procedimiento **Reset** o **Rewrite**.

En todo programa en el que se realiza una entrada o salida de datos a un dispositivo que no es el estándar, se han de incluir las órdenes correspondientes a la asignación de dispositivo, inicialización y cierre. El esquema siempre es el que puede verse en la figura 15.

```
Var
Identificador File of Tipodato
   .     .     .
Begin
   .     .     .
  Assign (identificador,
  FichString);
   .     .     .
  Rewrite (identificador);
  { o Reset (identificador);
  Close (identificador)
```

Fig. 15. En todo programa que se realice una entrada/salida de datos, se han de incluir estas órdenes.

• Archivos de tipos de datos definidos por el usuario

Un dispositivo de entrada/salida se puede definir como **File Of** para cualquier tipo de dato, incluyendo los definidos por el usuario. Son especialmente útiles los archivos del tipo registro, pues en ellos es posible almacenar fácilmente datos con la estructura interna requerida por el usuario. En las secciones anteriores se utilizaban los alumnos para ilustrar el uso de los registros. Ahora se continuará con el mismo ejemplo para mostrar lo conveniente que resulta utilizar archivos de datos con estructura de registro.

EL PROGRAMA "CREA"

Se va a suponer que se desea construir una base de datos en la que almacenar información respecto de los alumnos de un curso; siguiendo el mismo criterio utilizado anteriormente, se desean almacenar los datos de nombre, apellido y fecha de nacimiento de cada uno de ellos.

Lo primero que se va a considerar es un programa que cree la base de datos para uno de los cursos. Es decir, un programa en el que se cree el archivo que vaya a contener los datos y se escriba la información del primer alumno que formará parte de la base de datos. Este programa puede ser el denominado "Crea", que puede verse en la figura 16.

En él se empieza leyendo el nombre que va a tener el archivo. Algo así como "primero.dat", que servirá para que lo reconozca el sistema operativo. Una vez que se ha confirmado que los datos son correctos, se procede a asignar el identificador "carpeta" (un fichero de registros de alumnos) a ese archivo. Se abre el fichero por primera vez (**Rewrite**), se escriben los datos y se deja cerrado.

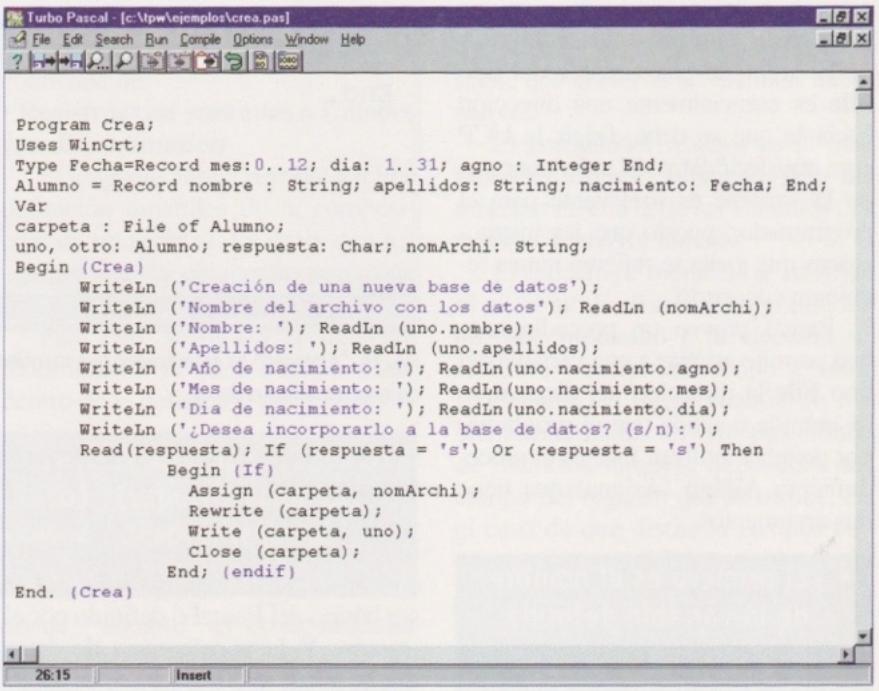


Fig. 16. En este programa se crea la base de datos con información de los alumnos.

Una vez ejecutado el programa "Crea", existirá en el disco duro el archivo que almacena la información de un sólo alumno.

No hay que intentar ver esa información con un editor de archivos ASCII, ya que no se podrá, pues los datos están almacenados con la misma secuencia de ceros y unos que se utilizan en la memoria del ordenador para guardar la información. Pero no hay que preocuparse; más adelante se estudiará el programa que consigue listar la información de la base de datos.

Antes de eso, se van a analizar los cambios que necesita este programa para ser una aplicación que permita añadir a la base de datos la información de un nuevo alumno.

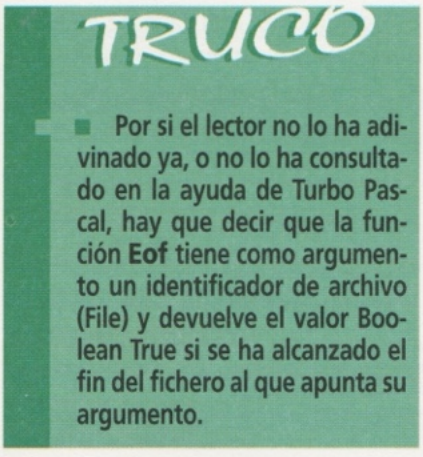
EL PROGRAMA "AUMENTA"

En la figura 17 puede verse el programa "Aumenta", que se encarga de añadir los datos del nuevo alumno. Los cambios que hay que hacer de "Crea" a "Aumenta" son muy pocos:

• Primero, se ha de abrir el fichero ya existente, por lo que la apertura se realiza con **Reset**.

• Segundo, se ha de colocar la cabeza de escritura sobre el disco, en la posición adecuada, una vez pasado el último "alumno" escrito. Para ello, se van leyendo datos de alumnos mientras no se llegue al final del archivo ("Eof(carpeta)").

Una vez alcanzado el final, se añade la información del nuevo. El programa "Aumenta" sólo sirve para añadir la información de un único alumno. Se dejará al lector, como ejercicio, la modificación del programa para que se puedan añadir varios como resultado de correr la aplicación "Aumenta".



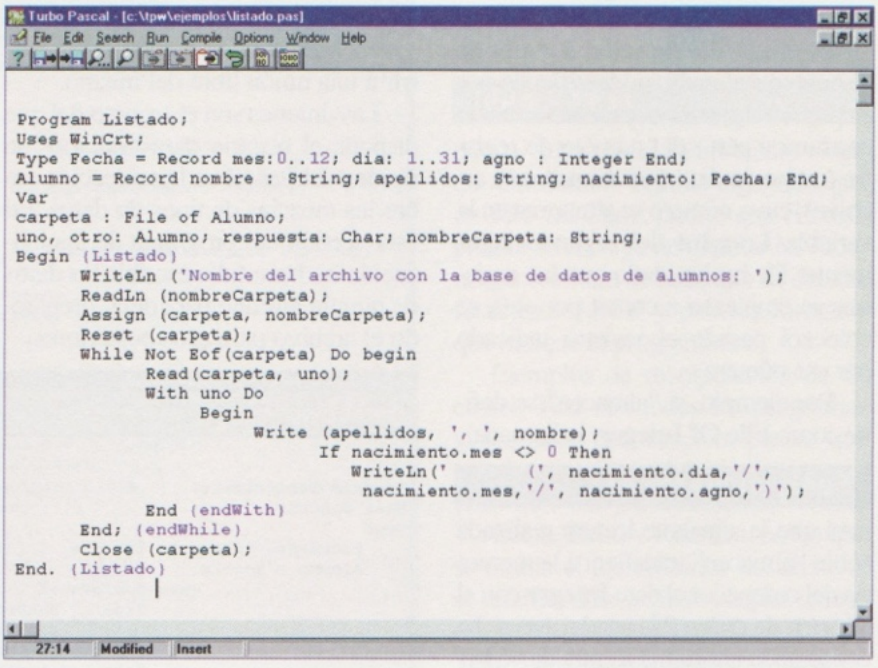


Fig. 18. Este programa se puede utilizar para obtener el listado con los datos de todos los alumnos almacenados en la base de datos.

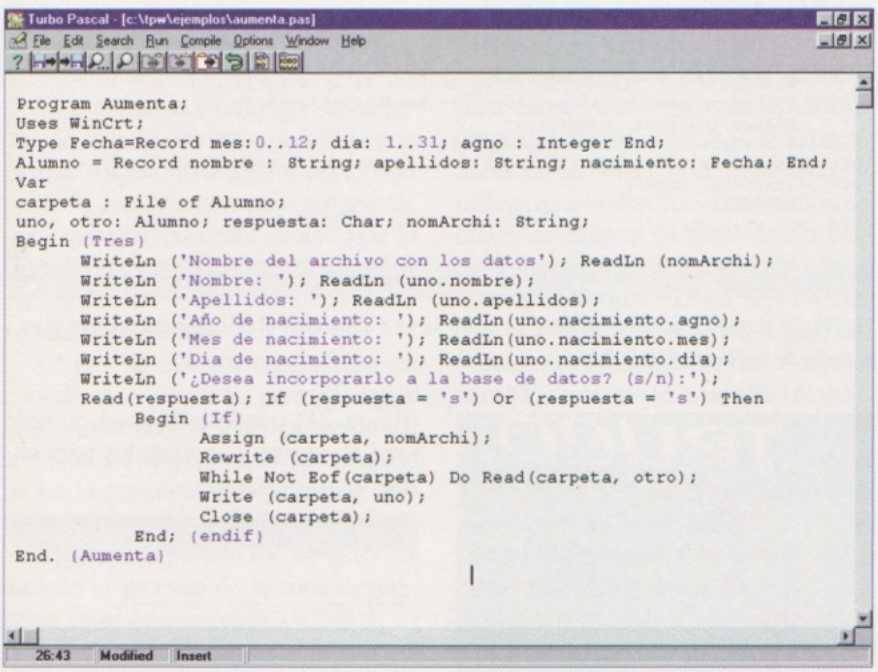


Fig. 17. "Aumenta" sólo permite añadir la información de un nuevo alumno a la base de datos.

EL PROGRAMA "LISTADO"

El listado de todos los alumnos incluidos en la base de datos se podría realizar mediante el programa "Listado", que puede verse en la figura 18.

Se recomienda al lector que vaya escribiendo todos estos programas de manejo de archivo, puesto que le será muy útil trabajar sobre ellos y producir modificaciones. Si durante su ejecución alguna vez aparece un mensaje de error ("Run time error"), lo más probable es que éste tenga su origen en el hecho de que se está intentando leer un archivo que no se ha creado previamente.

• Procesamiento aleatorio de Ficheros: "Seek"

Tanto en el programa "Aumenta" como en el "Listado" se está realizando un procesamiento secuencial de los archivos: se inicia la lectura por el primer registro y se procede hasta llegar al deseado o al final. El nombre "secuencial" también tiene que ver con la estructura lógica que se utiliza para guardar información en los ficheros. Ésta es, en esencia, muy parecida a la de los datos **Array**: una ristra de bytes uno detrás de otro.

En el caso del programa "Aumenta", este tipo de procesamiento ha obligado a transferir a la memoria el contenido de todos los registros para poder llegar hasta el último.

Dicho de otro modo, se ha estado leyendo toda la información del archivo para nada. Solamente para alcanzar el final del archivo. Pero es posible evitar ese trasiego inútil de información a la memoria RAM. Para ello, se ha de utilizar el procedimiento **Seek**, que tiene dos argumentos: el primero,

un indicador **File** y el segundo, una variable **LongInt**.

El efecto del procedimiento **Seek** es preparar el archivo identificado por el primer argumento para la lectura o escritura a partir del número de registro (o tipo de dato que almacena el archivo), cuyo número se almacena en la variable **LongInt** del segundo argumento. De hecho, dado que los registros se empiezan a contar por el 0, se colocará pasado el registro indicado por ese número.

Por ejemplo, si "almacen" se define como **File Of Integer**, la llamada

```
Seek (fichero, 23)
```

hará que la siguiente lectura realizada sobre "almacen" transfiera a la memoria del ordenador el dato **Integer** con el número de orden 24, con lo que se ha conseguido ahorrar la lectura de 46 bytes (2 por cada **Integer**) del fichero.

Al mismo tiempo, la instrucción

```
Seek (fichero, 0)
```

hará que esté disponible para la lectura el primer dato del archivo.

Con la posibilidad que brinda el procedimiento "Seek" se puede realizar lo que se llama **procesamiento aleatorio de ficheros**. A pesar del nombre, esto no tiene nada que ver con el azar, sino que implica leer o escribir en archivos en un orden distinto al secuencial.

EL PROGRAMA "INICIALISTA"

Para explicar la utilidad del **procesamiento aleatorio de ficheros**, se va a modificar el diseño de la base de datos de alumnos sugerida anteriormente. De entrada, se necesita tener en el archivo que almacena la base de datos alguna indicación del número de registros que hay almacenados.

En el programa "IniciaLista", que puede verse en la figura 19, se procede a inicializar la base de datos:

- se crea el archivo con el nombre indicado por el usuario y se escribe en el primer registro el **Integer** 0 para indicar que no hay todavía ningún alumno incluido.
- como "carpeta" se define un **File Of alumno**; para poder escribir un entero en el primer registro hay que recurrir a una unión libre del mismo.

Las uniones son el recurso del que dispone el usuario para "saltarse" la rigidez del Pascal en la protección sobre las mezclas de tipos de datos. En esta versión del programa de inicialización no hace falta escribir los datos de ningún alumno para dejar preparado el archivo para usos posteriores.

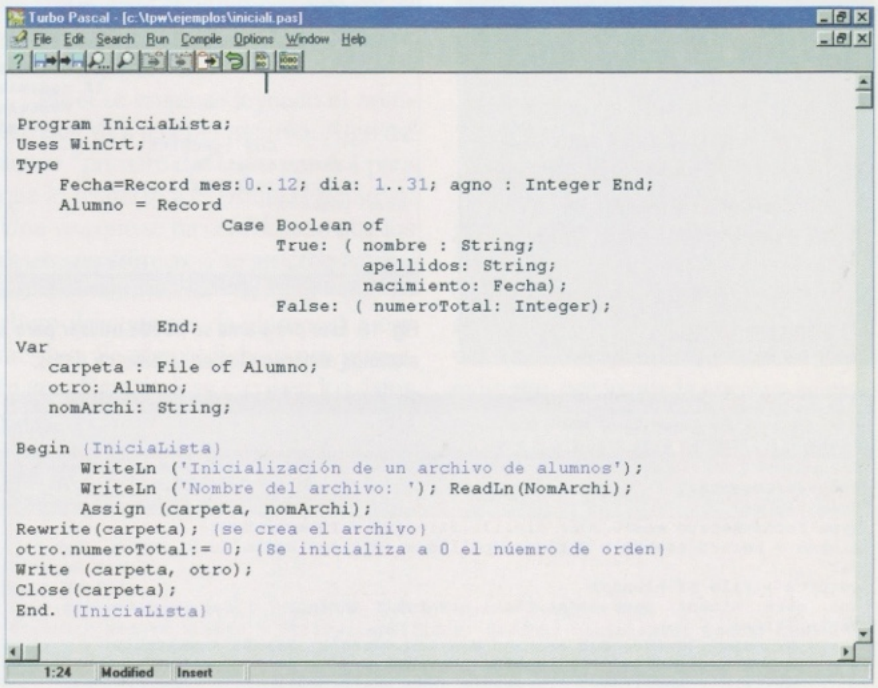


Fig. 19. En el nuevo diseño de la base de datos se va a almacenar al principio del archivo el número de registros guardados.

TRUCO

El programa "IniciaLista" podría ser mucho más corto porque, de hecho, el indicador "carpeta" podría haberse declarado directamente como File of Integer. La unión libre hace falta cuando en el mismo programa se van a leer o escribir, aparte del entero, los datos completos del registro alumno.

EL PROGRAMA "AUMENTALISTA"

El programa que puede añadir un alumno en la base de datos es el denominado "AumentaLista", que puede verse en la figura 20. Para llegar al final del fichero simplemente se saltan los registros indicados al principio y se escribe el nuevo. Después, se vuelve al principio para aumentar en 1 el contador de registros almacenados.

EL PROGRAMA "LISTALISTA"

Igualmente, el listado de alumnos que realiza el programa "ListaLista" (figura 21) puede utilizar el número total de alumnos registrados para extraer la información.

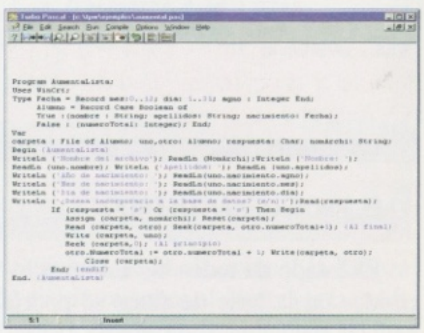


Fig. 20. Este programa también puede mejorarse para que el usuario controle el número de registros que quiera añadir.

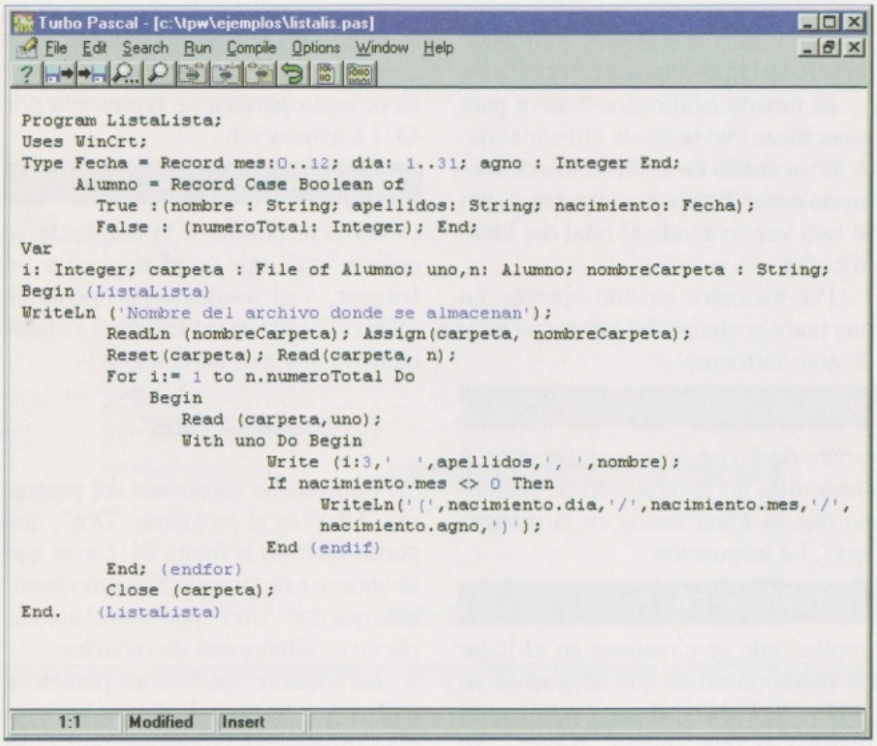


Fig. 21. Programa de listado en la nueva versión de la base de datos.

Con el nuevo diseño que se ha hecho de la base de datos, contando con el acceso aleatorio a la información del archivo, es fácil construir programas que permitan mantener actualizada la base de datos, es decir, borrar y modificar registros.

• **Ficheros de texto**

El uso de ficheros asociados a un tipo de dato concreto permite lograr que la información se almacene en los archivos del mismo modo que se escribe en la memoria del ordenador. Por tanto, no es necesario ninguna traducción para pasarlos a la memoria RAM durante el proceso de lectura, o viceversa, en el de escritura.

La ventaja de usar este tipo de archivos es la rapidez en la transferencia de información entre memoria RAM y dispositivos de almacenamiento.

Su inconveniente es la falta de compatibilidad, ya que para leer correctamente la secuencia de bits escrita por un programa Pascal en un fichero con tipo, es necesario u otro programa Pascal, o un programa muy sutil en otro lenguaje que tenga en cuenta la representación en binario de los datos.

En la actualidad, el modo estándar para transferir información es utilizar el código de caracteres ASCII almacenados en un byte, con el bit menos significativo a la derecha. Por tanto, si se quiere escribir en un archivo información que sea legible por la mayoría de las utilidades que existen hoy en día, hay que utilizar los datos **Char** del Pascal. Además, también es estándar el modo en el que se especifica el salto de línea.

Pascal suministra el tipo de fichero llamado **Text** (Texto) para definir dispositivos externos a los que se transfiere toda la información traducida a caracteres ASCII. Por ejemplo, los dispositivos estándar de entrada y salida (teclado y terminal) son un ejemplo de "Ficheros" **Text**.

Los procedimientos **ReadLn** y **WriteLn**, que no se podían utilizar con los ficheros con tipo, sí que es posible emplearlos con los ficheros de texto y son los que se encargan de gestionar los caracteres ASCII que indican el cambio de línea (se trata de la secuencia de los dos códigos 10 y 13 de la tabla ASCII).

Para declarar el identificador de un archivo del tipo **Text** se utiliza la sintaxis:

```
Iden : Text
```

donde "Iden" es el identificador elegido. Un fichero **Text** es algo más que un **File Of Char**, puesto que el compilador se encarga de traducir las órdenes de escritura de todo tipo de datos a caracteres ASCII y con los formatos que indique el programador.

Ejemplos de manipulación de ficheros **Text** son todos los programas vistos hasta esta sección, puesto que toda la entrada y salida de datos realizada con los procedimientos **Write**, **WriteLn**, **Read** y **ReadLn**, en los que se omitía el identificador del archivo, se referían a los archivos **Input** (entrada) y **Output** (salida) que son del tipo **Text**. Pero la desventaja principal de los archivos de este tipo es que no permiten acceso aleatorio; siempre han de procesarse de modo secuencial. Por tanto, quedan descartados para su uso en bases de datos.

PUNTEROS Y ASIGNACIÓN DINÁMICA DE MEMORIA

El nombre de la variable es el indicador que permite al compilador determinar a qué dato se hace referencia y generar código UCP para operar con él. Cada dato está almacenado en una posición de memoria y su contenido se puede modificar a través de la relación unívoca que existe entre el nombre de la variable y la posición que el dato ocupa en la memoria del ordenador. Ahora se va a considerar un programa tan sencillo como el mostrado en la figura 22:

```
Program Uno;
Var
a , b : Integer;
Begin { Uno }
{1}     ReadLn(a);
{2}     b := a;
{3}     WriteLn(a:4,' ',b:4);
End. { Uno }
```

Fig. 22. Programa Uno en el que se definen dos variables.

en el que se definen las variables "a" y "b" para almacenar datos del tipo **Integer**. En la instrucción {3} se especifica que el dato almacenado en la posición de memoria a la que hace referencia la variable "a", ha de almacenarse también en la posición de memoria a la que hace referencia la variable "b". En ningún momento se ha hecho explícito en la sintaxis del lenguaje que un dato esté identificado internamente por una posición de memoria.

Sin embargo, es muy útil poder manipular los datos realizando algoritmos en los que también intervenga la dirección de aquéllos. Para ello, Pascal permite que en los programas aparezcan variables que almacenen direcciones de otras variables. Estas variables que almacenan direcciones reciben el nombre de **punteros**.

Se especifica que una variable es un puntero a un tipo de dato con la ayuda del calificador ^ que se antepone a un identificador. La variable declarada como

```
^tipodato
```

es el puntero a un dato del tipo "tipodato". Por ejemplo, en el segmento de programa que puede verse en la figura 23:

```
Type
Fecha =
Record
dia : Integer;
mes : Integer;
End;
Var
a: Integer;
px : ^Real;
pa : ^Integer;
pHoy,pAyer : ^Fecha;
.    .    .
```

Fig. 23. Especificación de punteros a datos.

se está declarando que "px" es un puntero a un dato **Real**, "pa" un puntero a un **Integer**, y "pHoy" y "pAyer" punteros a registros de "Fecha". Es decir, en "px" se almacenarán direcciones de memoria de variables del tipo **Real**, en "pa" direcciones de enteros, y en "pHoy" la dirección donde empieza un registro del tipo "Fecha".

El mismo calificador ^ sirve para especificar que se desea utilizar el dato almacenado en la dirección de memoria especificada por el puntero, pero esta vez se añade al final del identificador.

Por ejemplo, cuando aparece en una parte posterior del programa citado anteriormente

```
px^
```

quiere decir que se hace referencia al **contenido** de la dirección de memoria que se almacenaba en el puntero "px". La asignación

```
pHoy^ := pAyer^;
```

implica que se almacene en el lugar de memoria en el que se guarda en "pHoy" el dato almacenado en el registro al que apunta "pAyer". Se trata de una asignación muy distinta a la que se realiza con

```
pHoy := pAyer;
```

pues en este caso se indica que se almacene en "pHoy" la dirección de memoria que se almacenaba en "pAyer". En el primer caso había trasiego de días y meses, mientras que en el segundo sólo se copia una dirección.

Para completar la herramienta es necesario un operador que sea capaz de extraer de una variable su dirección; este operador se representa por @. La asignación:

```
pa := @a;
```

es correcta porque a la izquierda se coloca "pa", que es el puntero a un **Integer**, y el resultado de operar @ sobre "a" (que es un **Integer**) es también el puntero a un **Integer**.

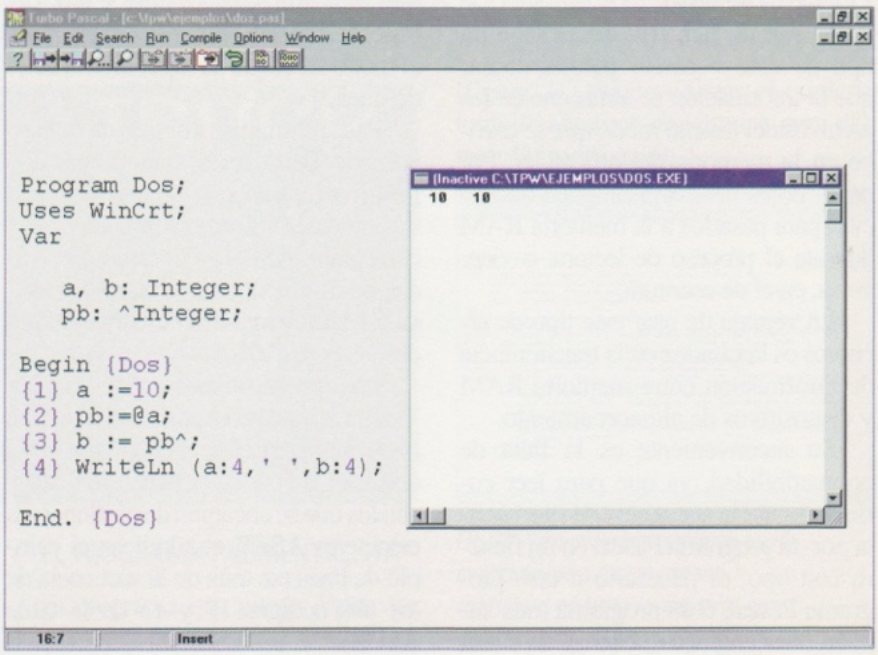
EL PROGRAMA "DOS"

Una versión sofisticada del programa "Uno" es el programa "Dos", que puede verse en la figura 24, con el que se obtiene exactamente el mismo resultado que con "Uno", pero con el uso explícito de direcciones de variables.

La variable "pb" es un puntero a un **Integer**, es decir, la dirección de un dato **Integer**; en la instrucción {2} se decide que ésta va a ser exactamente la dirección de la variable "a".

En la instrucción {3} se asigna a la variable "b" el dato que antes se había almacenado en la variable (puntero) "pb".

Fig. 24. En las instrucciones {2} y {3} se está haciendo en dos pasos la asignación a "b" del valor que almacena "a".



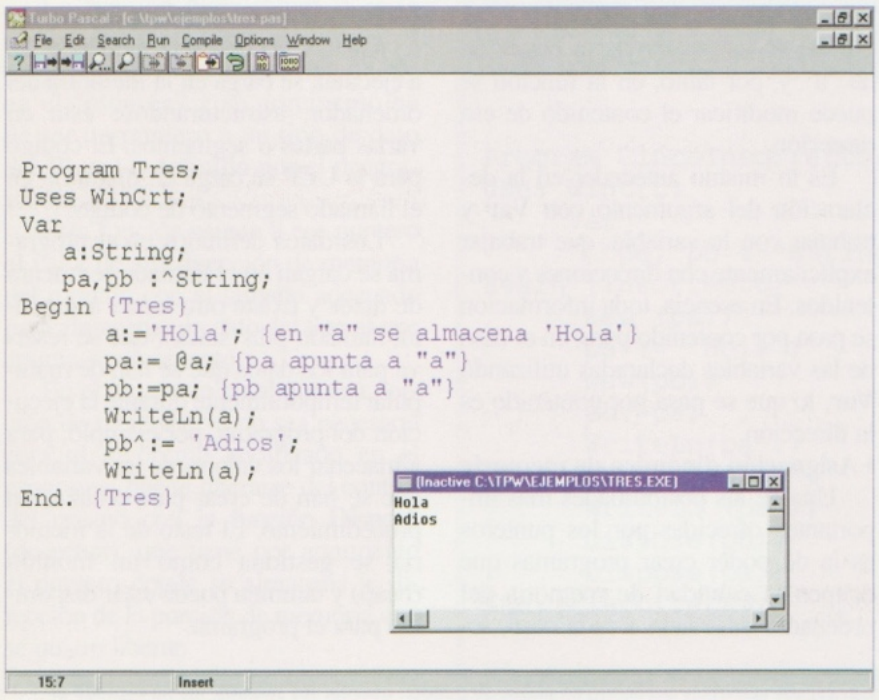
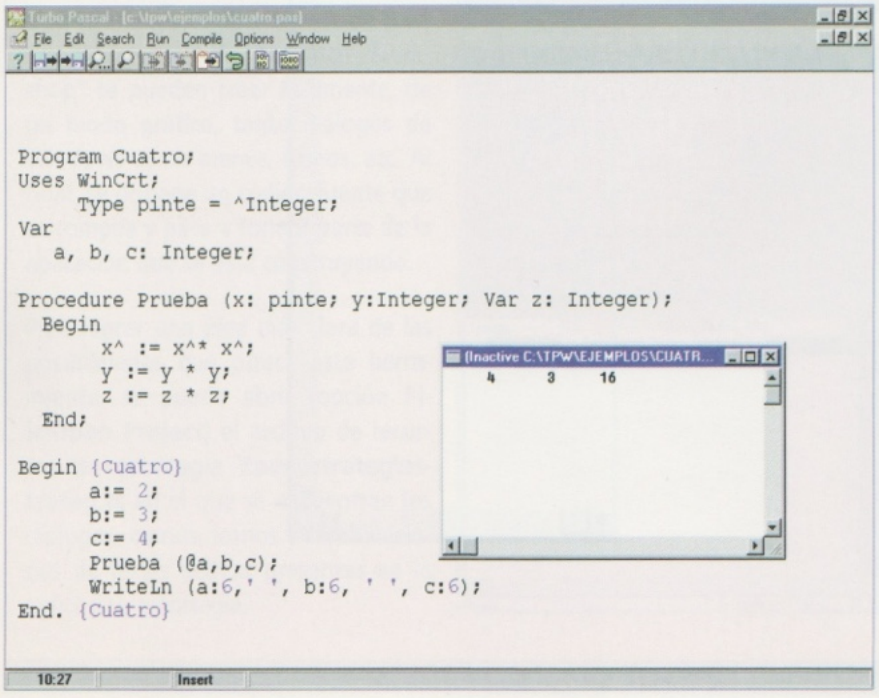


Fig. 25. Al modificar el contenido de "pb" se está procediendo a modificar el valor que almacena la variable "a".

Este dato almacenado en "b" será, pues, el mismo que el almacenado en "a". Las instrucciones {3} y {4} del programa "Dos" producen el mismo resultado que la {2} del programa "Uno", recorriendo un camino más largo y con una sintaxis menos clara. Pero si este tipo nuevo de dato se introduce en el lenguaje no es para complicarlo más, sino para aumentar sus posibilidades tal como se comentará a continuación.

Fig. 26. Aparte de las diferencias sintácticas, el modo en el que se comunica la información a las variables "x" y "z" del procedimiento "Prueba" es el mismo. Se pasa por contenido la dirección de la variable.



EL PROGRAMA "TRES"

Antes de seguir adelante, puede ser ilustrativo considerar el programa "Tres" que puede verse en la figura 25; en él se modifica indirectamente el valor del **String** "a" a través del contenido del puntero "pb".

LOS PUNTEROS

A estas alturas, al lector le habrán surgido ya un par de dudas. La primera es qué se guarda en las variables punteros, es decir, qué número almacenan. Pues bien, el Pascal está diseñado para que nunca se necesite saber este dato y no se haga nada que dependa de él. El programador sólo utilizará los punteros para manipular los datos a los que apuntan y evitará asignarles directamente valores numéricos. Generalmente se asigna a un puntero el contenido de otro. Hay una excepción: se trata del valor **Nil**, ya que no corresponde a un lugar de memoria y se utiliza para identificar que un puntero no apunta a ningún sitio. Las operaciones que se pueden realizar con los punteros son asignar y comparar por igualdad o desigualdad. Con estas comparaciones es posible discernir la equivalencia de direcciones. Sin embargo, no se pueden comparar los punteros con los operadores > y <.

La otra pregunta que habrá surgido, tiene que ver con el espacio de memoria que ocupan los punteros. La respuesta, por ahora, será que no importa; que para una misma máquina, el tamaño de memoria será el mismo para todos los punteros a cualquier tipo de dato. Se trata de una dirección y no importa la longitud de la variable para el tamaño de la dirección en donde empieza a estar situada.

Ahora que ya se ha entrado en el asunto de las direcciones de memoria de las variables, se podrá comprender el significado de los nombres que se dan a los dos modos de paso de argumentos a los procedimientos y funciones. Es decir, podrá comprenderse mejor porqué se denomina paso por referencia o dirección al argumento que va antecedido en la declaración por la palabra reservada **Var**. Como ya se dijo antes, en las variables que se pasan por referencia se está comunicando al módulo la dirección de la variable. Esta afirmación se puede ver claramente con el programa "Cuatro", de la figura 26, y la salida que produce.

El valor de la variable "b" no se modifica como resultado de la llamada a la función porque es un argumento pasado por contenido.

La variable "c" se modifica porque se pasa por referencia y la variable "a" también modifica su valor porque el argumento es la dirección de "a" y, por tanto, en la función se puede modificar el contenido de esa dirección.

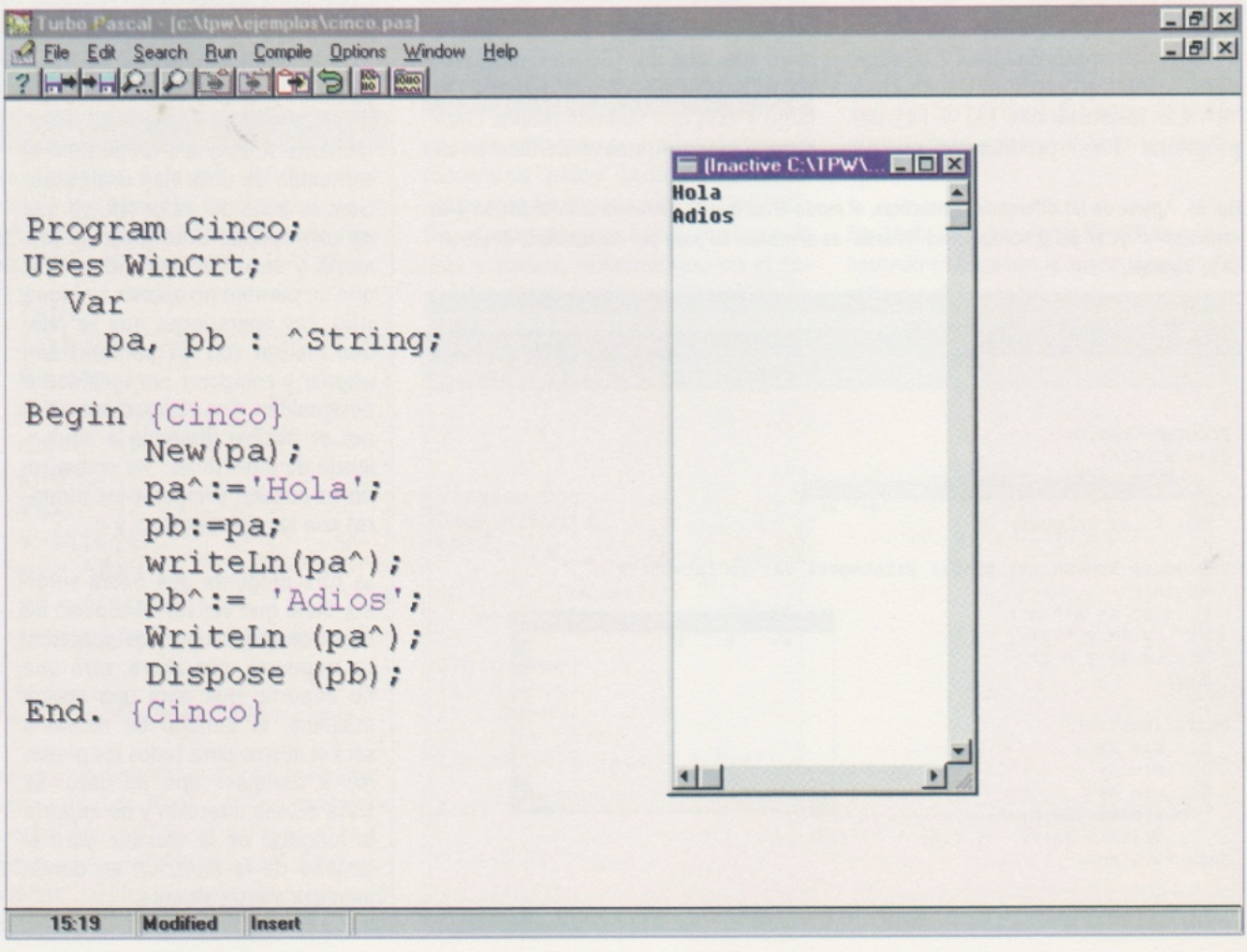
Es lo mismo anteceder en la declaración del argumento con **Var** y trabajar con la variable, que trabajar explícitamente con direcciones y contenidos. En esencia, toda información se pasa por contenido pero, en el caso de las variables declaradas utilizando **Var**, lo que se pasa por contenido es la dirección.

• **Asignación dinámica de memoria**

Una de las posibilidades más importantes ofrecidas por los punteros es la de poder crear programas que ocupen la cantidad de memoria del ordenador adecuada a cada caso, según se determine durante la ejecución de aquél. Cuando un programa se va a ejecutar, se carga en la memoria del ordenador, estructurándose ésta en varias partes o segmentos. El código para la UCP se carga en memoria en el llamado segmento de código.

Los datos definidos en el programa se cargan en el llamado segmento de datos y existe otro tercer segmento, llamado pila (stack), que se reserva para los datos que se han de manipular temporalmente durante la ejecución del programa, por ejemplo, para almacenar los valores de las variables que se han de crear para realizar un procedimiento. El resto de la memoria se gestiona como un montón (heap) y también puede estar disponible para el programa.

Fig. 27. Los 256 bytes del String en el que se pueden guardar los saludos, los ha captado el programa durante su ejecución.



La función **New** (Nuevo) es la que gestiona la captación para el programa de la memoria disponible en el "montón" y su argumento ha de ser un puntero a un tipo de dato del Pascal o definido por el programador.

Esta función asigna a ese puntero el valor de una dirección de memoria del montón con el tamaño adecuado para almacenar un dato del tipo al que apunta su argumento.

En el momento en el que esa porción de memoria ya no sea necesaria para el algoritmo codificado en el programa, puede liberarse del control del mismo con la función **Dispose** (desechar), que tiene por argumento el puntero donde se almacena la dirección de la porción de memoria que se quiere liberar.

A esta gestión de la memoria del ordenador durante la ejecución del programa se le llama **asignación dinámica de la memoria**.

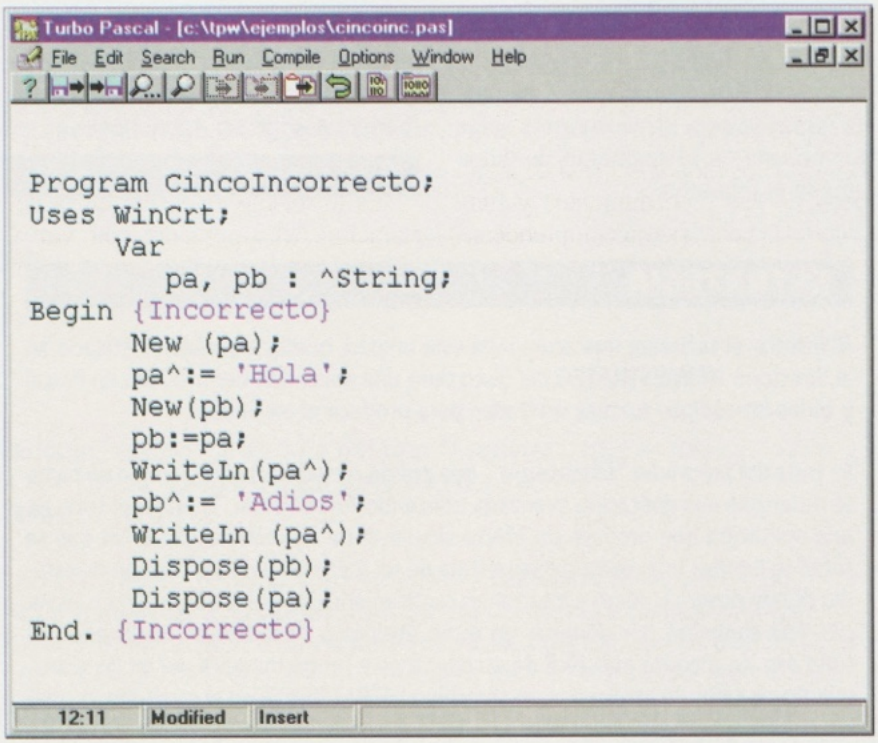


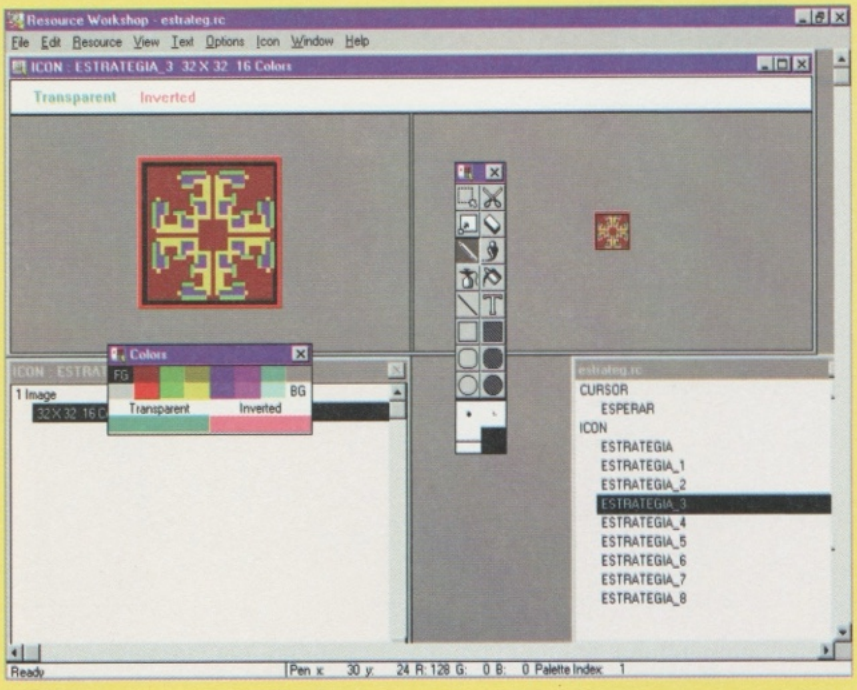
Fig. 28. Este programa dará un error durante su ejecución al intentar liberar una zona de memoria que ya no le corresponde al programa, puesto que la ha liberado con anterioridad.

RESOURCE WORKSHOP

En el grupo de programas de Turbo Pascal para Windows se ha instalado el elemento **Resource Workshop** (expresión inglesa cuya traducción podría ser "entorno de trabajo para recursos").

Se trata de la herramienta que ofrece Turbo Pascal para hacer más fácil construir el código de una aplicación para Windows. Con el "Resource Workshop" se pueden crear fácilmente, de un modo gráfico, tanto diálogos de Windows como menús, iconos, etc. Al final, se obtiene un código fuente que se compila y pasa a formar parte de la aplicación que se está construyendo.

Para lograr una idea más clara de las posibilidades que ofrece esta herramienta, se puede abrir (opción **File\Open Project**) el archivo de recursos de estrategia **\tpw\estrateg\estrateg.rc** en el que se encuentran los diálogos, menús, iconos y combinaciones de teclas activas presentes en la aplicación estrategia.



El programa "Cinco", que puede verse en la figura 27, es una modificación del "Tres" en el que la memoria utilizada para almacenar los saludos no está en el segmento de datos sino en el "montón".

Se reserva la memoria con **New** antes de almacenar ningún valor en esa dirección. La última instrucción hubiera podido ser, igualmente,

```
Dispose(pa);
```

porque ambos punteros almacenan la misma dirección de memoria.

Es más, el programa "CincoIncorrecto", que puede verse en la figura 28, está mal por varios motivos. Primero, el programa daría un error en la ejecución, ya que la última instrucción intenta liberar un lugar de memoria que ya ha sido liberado en la llamada anterior a **Dispose**.

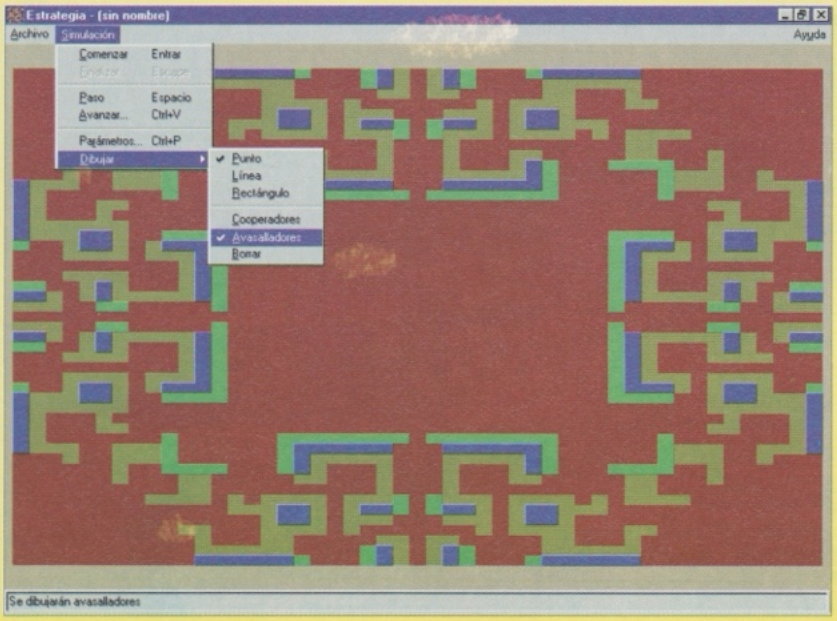
Por otra parte, es un ejemplo claro de mal uso de la gestión dinámica de memoria. La reservada para el programa mediante la tercera instrucción (256 bytes) no se puede liberar puesto que ya se ha perdido la pista de cuál era. Es una práctica poco recomendable escribir programas que, cuando acaben, dejan sin liberar memoria que reservaron del montón.

ESTRATEGIA

Al instalar el software que acompaña esta unidad, quedará también instalado en el directorio \TPW\ESTRATEG del disco duro una aplicación desarrollada en Pascal y todos los códigos fuentes utilizados para producir el ejecutable.

Se trata del programa "Estrategia", que puede considerarse un ejemplo de cómo se desarrolla una aplicación avanzada utilizando Turbo Pascal. También se trata de una aplicación que produce un efecto similar al de un calidoscopio en el que se suceden bonitas imágenes, o que se trata de un juego, o que es un campo de estudio donde observar cómo evolucionan caóticamente las poblaciones de dos especies que compiten por dominar un nicho ecológico. "Estrategia" es un poco de todo eso. La manera más fácil de acceder a este programa para ver su funcionamiento, es crear un acceso directo al archivo ESTRATEG.EXE en el escritorio, o crear un nuevo grupo de programas desde la carpeta Windows\Menú de inicio \Programas para depositar aquí el acceso directo anteriormente creado. En todo caso siempre podemos ejecutar el programa haciendo doble clic sobre el archivo ejecutable desde el explorador de Windows, o mediante el comando ejecutar del menú Inicio.

Los programas Pascal que componen esta aplicación se pueden encontrar en el directorio de la aplicación. Se trata de siete archivos de extensión .PAS que pueden abrirse con Turbo Pascal. De este modo, se accederá a más de 5000 líneas de código avanzado, pero bien estructurado y comentado. El programa principal se encuentra en el archivo "ESTRATEG.PAS" y en él aparecen las instrucciones necesarias para crear un nuevo ejecutable si se modifica el programa.



RESUMEN

Con esta unidad acaba el bloque del curso dedicado al aprendizaje de los conceptos básicos de programación utilizando el lenguaje Pascal. El lector ya estará preparado para escribir aplicaciones sencillas con Turbo Pascal pero, sobre todo, para comprender fácilmente el lenguaje de programación que ofrece el Sistema de Gestión de Bases de Datos Paradox y para avanzar rápidamente en el aprendizaje del lenguaje de programación C.

Sin embargo, conviene saber que Turbo Pascal es una herramienta con la que puede construirse cualquier tipo de aplicación para el entorno gráfico Windows, ya que incluye la extensión del Pascal con Programación Orientada a Objetos, con la que se pueden construir fácilmente aplicaciones Windows con sus objetos característicos.

En este curso se estudiarán todos esos conceptos avanzados utilizando el lenguaje C, pero hay que saber que también están disponibles siguiendo el camino del Pascal.

CUESTIONARIO AUTOEVALUATIVO

1 - Una enumeración **No** se puede utilizar para asignar indicadores a ¿qué secuencia de datos?

- ❑ A - **Byte.**
- ❑ B - **Integer.**
- ❑ C - **Real.**
- ❑ D - **Char.**

2 - Si la variable "cocina" se declara del tipo "Esquinas", que es una enumeración con cuatro componentes, ¿qué sucederá con la instrucción: **ReadLn**(cocina).?

- ❑ A - Es incorrecta sintácticamente y el compilador detectará la incoherencia.
- ❑ B - Dará lugar a un error si el usuario suministra un número con decimales.
- ❑ C - Dará lugar a un error si el usuario suministra un número mayor que 3.
- ❑ D - Dará lugar a un error si el usuario suministra un número mayor que 4.

3 - En un programa se declara el registro "cosas" como **Type** cosas = **Record** a: **Integer**; b: **Real**; c: **Boolean** End; y las variables **Var** una,otra : Cosas. ¿Cuánto ocupará en memoria cada una de las variables?

- ❑ A - 8 bytes.
- ❑ B - 9 bytes.
- ❑ C - 4 bytes.
- ❑ D - 8 bytes y 1 bit.

4 - ¿Es correcta la asignación una.b := otra.a?

- ❑ A - Es incorrecta sintácticamente y el compilador avisará de la incoherencia.
- ❑ B - No y el error se detectará en ejecución.
- ❑ C - Puede ser incorrecta dependiendo del valor de los números que se intercambian.
- ❑ D - Sí.

5 - ¿Una unión libre se puede utilizar para declarar una variación del contenido del campo de un registro?

- ❑ A - Sí, sea cual sea el número de orden del campo dentro del registro.
- ❑ B - Sí, cuando se trate del último campo del registro.
- ❑ C - Sí, cuando las distintas variantes ocupen el mismo espacio en la memoria del ordenador.
- ❑ D - Sí, cuando se trate del primer campo del registro.

6 - ¿Qué ocurre si se abre con **Rewrite** un archivo que ya existía?

- ❑ A - Se pierde toda la información de la copia existente antes de ejecutarse el programa.
- ❑ B - Se reescribe una copia del archivo en un backup.
- ❑ C - La nueva información se añade al final de la ya existente.
- ❑ D - La vieja información se escribe al final de la nueva.

7 - El acceso aleatorio a un fichero:

- ❑ A - Está gobernado por la función **Random** que genera secuencias de números aleatorios.
- ❑ B - Permite fraccionar un archivo en un número aleatorio de partes.
- ❑ C - Evita el acceso secuencial al contenido del archivo.
- ❑ D - Debe siempre inicializarse con una llamada a **Randomize**.

8 - Cuando las variables "ico" e "iso" se declaran **Var** ico : **^Real**; iso: **^Byte**; ¿cuánta memoria ocupará cada una?

- ❑ A - "ico" ocupará en memoria 6 veces más que "iso".
- ❑ B - "iso" ocupará en memoria 6 veces más que "ico".
- ❑ C - Las dos ocuparán lo mismo en la memoria.
- ❑ D - El tamaño relativo que ocupen en memoria dependerá de la dirección que almacenen.

9 - Si en un programa, a la declaración de variables de la cuestión anterior se añade la nueva declaración perico : **Real**; ¿la siguiente asignación será correcta o incorrecta? iso := @perico;

- ❑ A - correcta.
- ❑ B - incorrecta y el error se detectará en ejecución.
- ❑ C - incorrecta y el error se detectará en compilación.
- ❑ D - correcta si la dirección de la variable "perico" es un número entero.

10-Siguiendo con el mismo programa de la cuestión anterior, la asignación ico := @perico + 3; ¿es correcta o incorrecta?

- ❑ A - Incorrecta y el error se detectará en compilación.
- ❑ B - Incorrecta y el error se detectará en ejecución.
- ❑ C - Correcta.
- ❑ D - Correcta si la dirección de la variable "perico" no está en el límite de un segmento de memoria.

Respuestas: 1 C - 2 A - 3 B - 4 D - 5 B - 6 A - 7 C - 8 C - 9 C - 10 A